SCTE ISBE CABLE-TEC EXPO '16
SEPTEMBER 26-29 PHILADELPHIA

SCTE Society of Cable Telecommunications Engineers
ISBE International Society of Broadband Experts

The Open Encoder Project

Moving Video Processing to the Cloud with Containers, Standard APIs, and Common Alarms

A Technical Paper Prepared for SCTE/ISBE by

Scott M Davis
Principal Architect
Charter Communications
Broomfield, CO
Scott.davis1@Charter.com

Table of Contents

List of Figures

Introduction

Many cable companies have an operational and organizational goal to move to the cloud with more of our internal services and processes. For video products themselves (encoders, multiplexors, decoders, IRDs) Charter Communications is heavily invested in chassis based solutions from several vendors (carrying the logo for each vendor). The majority of our encoders are a vendor produced chassis, the exception being some of our ABR and VOD reprocessing stacks (some of these encoding solutions are Windows based applications, with some having an option to go Linux applications in 2016). While we have successfully moved a great many processes and administrative functions into the cloud domain, moving video processing to the cloud has lagged, thus we have kicked off a plan to move in that direction through the Open Encoder project.

The Open Encoder project is not open in the sense of Open Source projects (although we are not declining that as an option), but rather an attempt to create a public workflow for video and audio streams where the interfaces are publically available, controls are well defined (and again, open for people to review and contribute), and unlocked to single vendor or implementer control. We would like to open it to any vendor to come in and assert their blocks into without excluding other vendors. At this time, we are exploring a fully defined object (specifically an encoding unit), but in the future, it may be possible to decompose the individual functions inside that unit further.

This paper looks at the pros and cons of using the cloud for encoding linear streams. Many of options presented in the press or online examine the ease of the tools around getting into the cloud (either focusing on the orchestration layer or the new features that make operating the environment easier). Very little seems to talk about the hurdles to overcome for the video product itself. The project we are currently working on is focused on what operational issues we may face, what changes to our workflow we need to consider, and (in the longer term) what the cost benefit analysis tells us (is it better, cheaper, easier to operate).

As a disclaimer, we have not progressed anywhere near as far as we would have liked. Much of the work that has been completed ended up being done inside our partner vendors labs with small segments done in our labs. Where we would like to be looking at this from the inside, there are larger parts that continue to be investigated.

Motivation

At this time, Content Operations at Charter Communications operates over 4000 channels of linear encoding in multiple data centers in a mix of ABR and classic modes. We also operate several different VOD encoding farms as well as ingesting pre-encoded products. No two of these systems operate using the same control protocols or APIs. While some of these systems have RESTful APIs, most are UI driven which makes independent automation of such a vastly diverse system difficult. Modification of the environment by install and removal chassis has become enormously onerous.

Nearly all of the encoding we operate is based on custom hardware from the vendor. Purchase and installation of a new model of a chassis can be set back significantly by the organizational processes in place. Managing large scale deployments of new firmware can take weeks to months, and it limits our ability to be agile against the needs of the business. Many promises have been made in the telecom industry about the increased benefits to moving to a cloud environment for video, but the rapid

deployment on common hardware (that is, server based solutions) and software updates alone is enough of a motivation for us to examine the options.

Additionally, there are rapid changes in new standards would be easier to manage in a pure software world. It makes sense for us to test HEVC, 10 bit, HDR, WCG, and HFR in software first, and then determine which products and standards we will support. While we believe there will always be some aspects that are best handled in hardware based products, working specific technologies in the cloud would let us migrate where the business needs dictate rapidly.

Goals

Our goal with the Open Encoder project is to implement encoding purely in software where possible. We also would like to begin to standardize our APIs and alarms across our products, and stand up a common means to collect alarms that are more rational and meaningful from an operations point of view. The Open Encoder project is meant to decrease the manpower and time to upgrade software and move or deploy services while improving the operational fitness of the processes used to modify or manage those same services.

We chose to investigate the implementations and the open source tools that we could use for moving content in and out of the cloud, what operational issues we faced around failover of services, resource tracking for redundancy, and what orchestration issues we would have to resolve. While we continue to work towards a cloud solution for VOD (for managing bursts of content), this paper focuses on the move of linear content to the cloud. We expect many of the lessons we learn to help us adapt and adopt common solutions for linear and VOD where they present themselves.

A large part of this effort is centered on operational consideration: what tools did we need to find or develop to ensure we can monitor flows, what information should we gather for analysis, and what interfaces would we need to define to get content from the existing video domain into the cloud?

Finally, what could we learn from existing implementations (either complete solutions or partial solutions) to make our architecture model work faster or sooner was of interest (thus surveying the existing offerings from our vendors was part of the project). All of this needed to be measured against the human effort to run the system of the future versus what is done today, and we needed to ensure that the use of the cloud did not negatively impact the output of the encoder in terms of video quality nor inappropriately change the costs of managing content.

Architecture

The first part of the project was determining how we would approach the cloud, either as an application based project or a container based project. Based on our experience with running standalone applications on Windows and Linux servers, and working along with our vendor partners, it became evident that the use of containers (specifically Docker) allowed for more flexibility for deploying in the cloud on arbitrary hardware, as this allowed us to move containers around quickly in the environment, rather than tuning the application against a specific hardware/OS combination. Since several potential partners had already moved in this direction, we ended up working with multiple vendors to test functions in their labs and ours, aggregating the results in to this paper. While many operators and vendors had decided to do container optimizations, there was a preponderance of vendors that had approached this as a cable product rather than as an IT option. As a result, there was a strong desire on some vendors' behalf to tie the whole

architecture together as a monolithic offering rather than as flexible modules we could interoperate. This "all in one bite" approach is orthogonal to our goals of swapping one for another in the individual modules to get a best for best approach. We as an industry will have to work hard to ensure we have the flexible required to move specific functions between vendors as seamlessly as possible.

Since we desired to test cloud encoding, we needed to have a fairly standard means to test different containers without having other aspects of the Docker cloud and container orchestration get in the way. In each of our tests a structure very similar to Figure 1 where microservices in containers were managed by an open source process set (often Mesos and Marathon), then allowing that to be run on a virtualization platform (such as Open Stack). We had originally looked at running Docker directly on Linux on bare metal managed via Ironic, but the networking was vastly improved by introducing Open Stack. We have attempted to implement this on generic hardware as much as possible, since we wanted to understand how to generically use cloud computation for encoding. While we are not absolutely mandating common compute at this time, it seems orthogonal to get into the cloud but then require very specific modules or blades are part of the solution.

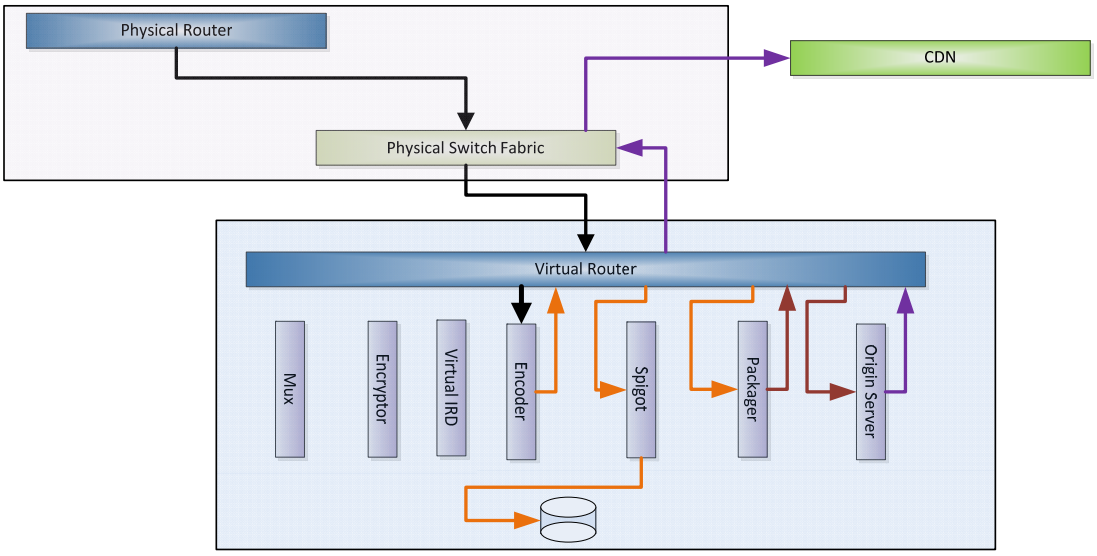

Figure 1 - Simple Architecture

In this first part of the project, we loaded control software on virtual machines as well as the containers just to get the project moving (installing the vendors control plane as well as their containers). While this paper focused on encoding, from Figure 1 you can see we reference other products than encoders, which we continue to work with various vendors on as well.

Orchestration of applications/containers in a cloud environment is the topic of multiple seminars, conferences, and other papers. There are multiple options available to us to use to support container migration, host discovery, and various other functional controls important in operations. At this time,

we've looked only as far as it takes for us to get started and testing in the environment, and have targeted Marathon and Mesos to allow us to run a Docker environment and manage the container roll out. Ideally, we will do a more complete analysis in parallel with encoder software development to refine these choices and to look at a more complete toolset. The tools available in this domain continue to increase in number and maturity. Even the choice to use Docker as a container system itself will be reviewed (to look at options such as Rocket and Drawbridge) as the project continues.

Progress

The testing so far has taking place on a blend of cloud environments with a mix of E3 and E5 Intel processors, without any specific hardware modules such as accelerators or GPUs. We found that for HD channels for ABR, we were able to run them successfully in a relatively small footprint. There is some variance, but we expect that to continue as different implementations take different approaches. We found that a single HD channel took roughly 10 E3 cores and 8 GByte of memory to run with 7 profiles at this time, which is close to what we would use for a standalone blade server to run a single channel today.

As mentioned above, each of the implementations we have worked with required a mux in front of it to translate from UDP multicast to RTP flows. For this, when working with vendors in their environments, we allowed them to select the mux, and have had no issues with any of the muxes involved in this translation to ingress feeds into the cloud. As we have a mix of implementations where the vendor has an internal packager also in the cloud or no packager (running a simple receiver to show that we are generating feeds), we have not worked the second half of the process to go from RTP with FEC back to multicast, but having done this before, we expect this to be trivial.

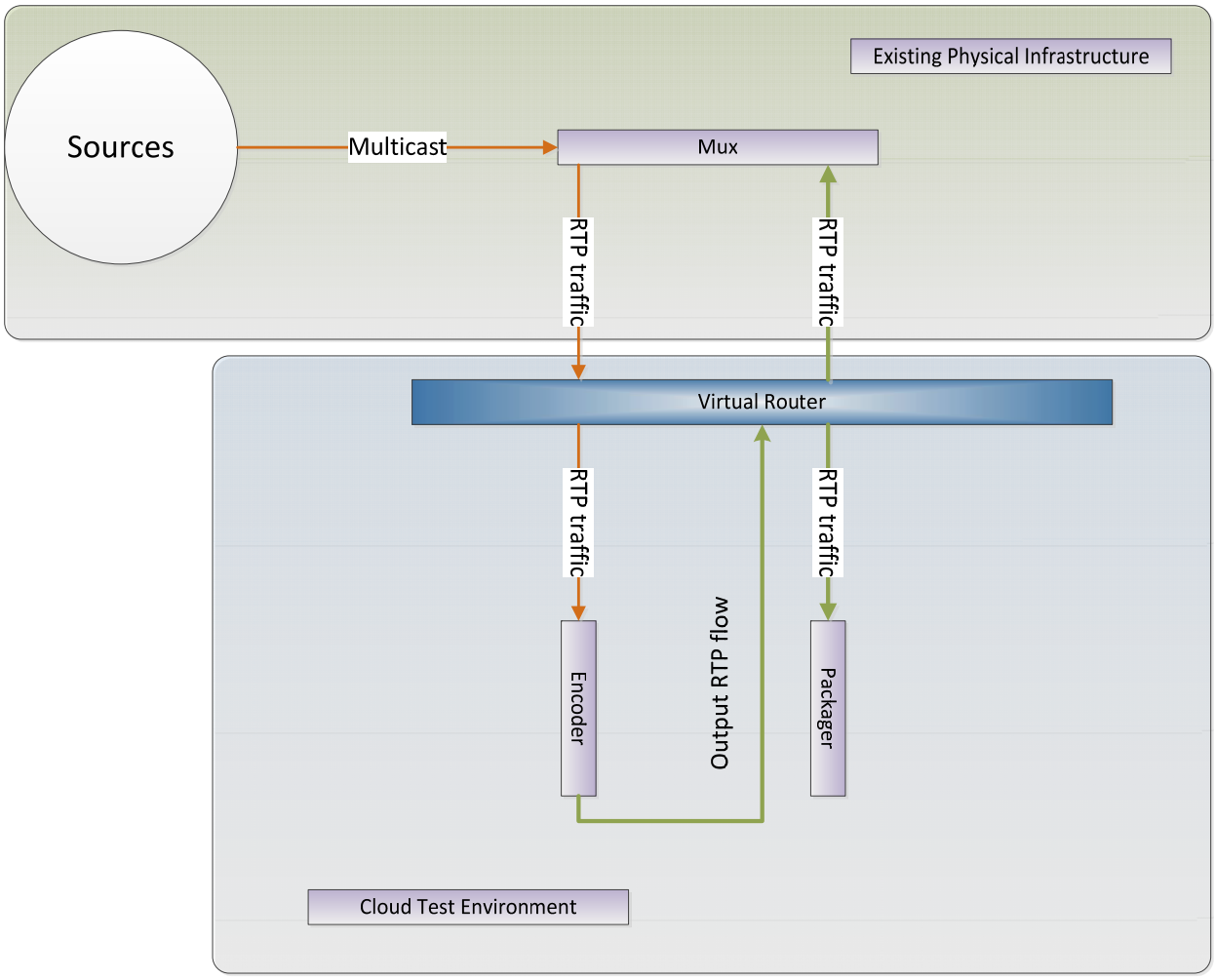

Figure 2 - Example Test Environment

For the setup (
Figure 2), the general design was using a standard MPEG2 transport stream received by a mux in multicast then using RTP to forward the traffic to a specified encoder in the cloud via it's IP address (for the data interface). The cloud was front ended by a virtual router receiving the traffic and forwarding it onto the unit under test. Typical testing required that a management unit be operative in the cloud to set up the encoder while communicating with the cloud orchestration layer to ensure that the proper encoder profile was set up on the cloud instance to receive and encode the traffic. Where we had a packager to forward the traffic to, we set that up in the cloud as well. If we did not have that, we routed the traffic via RTP back out of the cloud to a mux that could confirm good output.

Pros for the Cloud

As we expected, we found no change in the software based encoding when we used containers. We do see a minor impact on the computational resources used to run virtual machines in Open Stack, but that is offset by not needing retune the application against the individual hardware (N.B.: as the encoding team, we did not need to tune, but we did see that the cloud operations teams we worked with had to tune Docker to the hardware, so the process is not without its deployment overhead). We were able to test various implementations on non-specific hardware with little to no effort in getting the relevant video applications to run.

One aspect we expected to be very difficult that turned out to be relatively painless was service failure recovery. If we had a service failure (e.g. an error caused our container to fail, an issue caused the encoder in the container to fail, etc.), it was trivial for the systems we tested to turn down the existing container, spin up a new one, assert the service template to it, and start encoding. This is where we believe the cloud implementation shines best. There is no need to wait for a team member to move services manually, but rather the cloud implementations we worked with provided robust and fast recovery. Making this more transparent and appropriately reported will be part of the ongoing services orchestration layer that we will discuss later.

When we ran into issues in the cloud implementations for video products, one stellar positive was that we could leave an instance up and running in a “broken” state while we spun up a new instance to continue testing on. In a world where the expense and time required to keep spares on site for such functionality has always been prohibitive, this was enormously simple and lightweight.

As we needed defined IP addresses for our first implementation, we needed a means to assign and use IP addresses from outside the cloud environment (that is, IP addresses already assigned to physical devices to then migrate into the cloud). The cloud operations teams had a very simple “Bring Your Own IP” process that allowed us to move addresses from hardware to cloud. As a result, we had little in the way of issues of defining the endpoints for taking existing content into the cloud.

We had concern about increased latency, but other than adding a processing stage for the mux to translate to RTP, we’ve found little to no difference between the standalone software solutions we are using today and the cloud solution. We intend to take more significant effort to prove this out, but there have been positive results so far. We intend to take the processing stage before the current mux to RTP layer and move the RTP transmission to the location of the current multicast output and see if we can maintain our existing latency.

One other aspect we have begun to see is the speed of changes. Typical installations that we operate today involve testing monolithic software releases incorporating more changes than what are typically asked for (as vendors try to manage requests from multiple customers, they integrate changes that may have unintended consequences). In testing several of the vendor solutions, getting a new release of code to fix a specific bug was much simpler to narrow down to a very small number of lines of code, eliminating some of the “unintended feature” additions we have dealt with. In cooperation with other companies looking at similar architectures, we have found the ability to change code in a “micro release” paradigm vastly decreases the effort to move fixes into production code.

Cons for the Cloud

Networking (in terms of routing video) has turned out to be the hardest part of the process we have dealt with. As we decided earlier to do linear as a starting point rather than VOD, we spent a great deal of time trying to match our current practice of running multicast via UDP to an encoder. We quickly learned that multicast in the cloud was a nonstarter (N.B.: we do know that various folks are working to take multicast into the cloud; however, we are on a longer term plan to eliminate multicast in our environment for encoding, so there is a natural move to a unicast environment anyways). As we worked with a few of our vendors, even using unicast UDP packets provided us with challenges. In the earliest work, we found that we could not guarantee in order packet delivery (one of the key requirements we have had for the longest time to ensure the encoders would work). As a result, we are working to use RTP with FEC to ensure packets are delivered in order to the input queue to the encoder. This has required us to modify our environment to put a mux in between the physical network and the virtual router in the cloud to translate for us. While this is workable, we would like to move to a native flow that matches our existing workflow. We will continue to see if we can go to a simple UDP unicast flow in the future. As we consider a move to IPv6 (in the future), we will have to review this option to see what we need to support for the video flow.

One immediate concern coming from the operations side is the need to monitor the output of the encoder in the unicast domain and in the cloud. Legacy teams use a variety of probes to monitor the transport stream layer as well as the elementary streams encapsulated inside the transport. A certain amount of these probes are becoming aware of the underlying network and can report statistics on packet delay and round trip time. Inside the cloud, the hooks that we would expect aren't as readily available. While we expect many vendors to create these tools, we have started on a project called SPIGOT which will enable us to create a new container and instantiate it as an endpoint for delivery of services for recording the output for later analysis. In its first phase, it is nothing more than a transport stream recorder; in the future, we'd like it to become aware of tags inside the stream to determine beginning and ending of programs, blackout events, ad avails, and other conditioning so that it could be used for live to VOD applications, advanced services delivery, and continuity checking in real time. As we refine this, we intend to offer the source code for vendors to integrate the recording function into their software to natively record what the encoder believes it is delivering.

One area of concern that we have exposed is the rapid changes in the options for workflow management at the IT level. Hardly a day went by during any part of this work where some new tool or options were presented to the Docker framework or the virtualization for hardware. While having more options is usually desirable, there are nearly too many options and the speed at which the tools are maturing is a bit alarming. It seems likely that in the near future, we will lock down our container management and bypass the use of new tools and focus purely on the video functions, then play catch up on the IT level orchestration tools later.

Finally, we have to acknowledge the impact of the environment itself on the software we are trying to test. Management of the specific environment changes (such as modified libraries) needs very intense scrutiny. During testing of an aspect for our work in terms of ad insertion into the workflow, a simple change in the Open Stack environment ended up causing dropped packets, failure to deliver acknowledgements to messages, and some fragmentation of packets interfered with the testing for much of a month. Only once we found that a seemingly unrelated library was changed and reverted did we find the culprit. Ideally, as people refine their containers and incorporate their required versions of different code bases into their releases, this effect should decrease, but it still requires much vigilance.

Further Work

Early on, we discovered we realized that while the configuration of machines and containers was well understood, the ability to manage individual channels, their sources, destinations, and profiles was not a process that many vendors were ready to execute. Services management is mostly by hand for now as we turn up the containers and test them. Fortunately, the content operations team had two projects in flight that would assist: CAPO/Safehouse and VOD Controls. CAPO and Safehouse are tools that the operations software support team had created in the past for managing our existing linear processes, while the VOD controls project was working on when to create a service and to enable it. Together, these functions created the ability for a member of the staff to define a channel, its endpoints, the network graph for routing, and the ability to determine when we needed to create a new encoder for a service when one was not ready. While this is in its infancy, early indications are that adding cloud services to CAPO/Safehouse should be fairly trivial and building the hooks into it to call for a new container to be created for encoding live or linear are in flight.

We are planning on using the in house Safehouse, CAPO, and Blueprint processes to manage where a service goes once instantiations of the encoder are complete and ready for use. For launching a service, today's implementation requires that we have a standalone encoder with an empty "slot" (place to encode) on it, a knowledge of the input source, its video, audio, and data elements, and the output results. The operator creates a template for implementation, names the service, finds its internal stream number (that is, the number that we use for identifying to ourselves which stream it is), and creates a run book. For us to move this to the cloud encoder, rather than finding an open slot, we'll query the management segment of the cloud to determine if we have resources (or an extant encoder not in use for some reason), allocate the same template information to it, wait for it to join the input and create the output, and mark it in use as we would a physical encoder.

In our first pass of this process, this has become the most heavily rethought part of the work. Each vendor shows up with a different API, which requires time for our operations software team to program against, which delays our ability to launch. Each of these then requires time for us to test and verify that the encoder works as expected, returns a meaningful result, and allows us to manage it as desired. As a result, we have kicked off what we call The Open API Project.

The goals of the Open API Project are to define a common API set for functions that are common across a variety of MPEG aware applications that we can deploy in the cloud. For example, demuxing, PID selection, output IP addresses, input buffer length, and a plethora of other functions are fairly common across a broad range of video modules we could containerize. We strive to create a common framework for MPEG aware products Charter Communications can implement or partner with companies to implement. Since it is onerous to recreate automation for each of these based on vendors implementations, this limits our ability to integrate new vendors into our workflow. Ergo, we are considering a template of RESTful API calls that we would require for each vendor to implement. Thus, we would make implementation for automation the vendor's task, not ours (much as what was done with the early work with the test harness for switched digital video). The operations tools team has already started standardizing on the S3 API from Amazon to manage storage for the VOD workflow. Since each of the storage vendors they use have agreed to this, we have no need to re-write or re-implement any part of the VOD workflow that requires additional storage options. While we have considered working with AIMS on extending the reach of their documentation, for now, we believe we will handle this internally and publish for common consumption. By providing a Charter Communications standard API specification, we hope to ensure that most common functions for input and output, system configuration,

and service application are the same irrespective of the vendor we purchase the solution from. This should allow Charter Communications to we have more flexibility to choose solutions against need versus refactoring existing installation.

We also consider this an excellent opportunity to update alarming in the encoder module to a newer methodology. While SNMP has worked, it can be overly lightweight and the alarms themselves can be arcane and indecipherable. We have proposed a move to a JSON trap model where when the encoder is instantiated, part of the startup of the process passes on a listener IP and port number. When alarms, alerts, or other notifications occur, the encoder module would be expected to pass a JSON object to the IP address/port pair so that more meaningful information can be passed to the operator. An early instantiation of this process is being moved to our existing platforms for refinement. Some folks have suggested a polling model, however, polling hundreds (or thousands) of instances takes too long to cycle through in the linear world, so getting a modern trap engine is critical.

Conclusions

As this work is in many ways only barely started, we expect to find more issues such as the need to add an RTP layer and hurdles to jump such as creating the SPIGOT tool. At this time, we've concluded that our ability to run containers for encoding does not negatively impact the video quality of the encoding we do today, nor does it appear to increase our latency when compared to software based solutions. We have also determined that the containerization on top of virtual machines does not significantly affect our capacity when compared to existing software solutions. We must however do the analysis against existing hardware solutions and determine if we are close enough or ahead of them to make the switch to the cloud.

We have begun to dig into the onion layers that we'll need to unravel to ensure we can operate the system reliably. We've also begun to understand the complexity of the operation of real time encoding with an eye to the orchestration layer, which we believe is the hardest aspect of the project and likely the longest pole in the tent to make this work. While the software tools team in content operations has developed several segments required to make this work, much more effort must be expended. Completing a document for the open API is critical for this to succeed.

One large concern is the effort some vendors are exerting to create a monolithic solution where all the individual components are only exposed to each other. That is core to our ability to swap out components for best of breed implementations. It is imperative that we enable each vendor to have the ability to be swapped in. Again, this points to a need for a single comprehensive workflow controller with public APIs that allow a minimum of integration time.

Abbreviations

ABR	Adaptive Bit Rate
API	Application Program Interface
FEC	Forward Error Correction
GPU	Graphics Processing Unit
HEVC	High Efficiency Video Coding
HD	High Definition
HDR	High Dynamic Range
HFR	High Frame Rate
IP	Internet Protocol
JSON	JavaScript Object Notation
MPEG	Moving Picture Experts Group
REST	Representational State Transfer
RTP	Real-time Transport Protocol
SNMP	Simple Network Management Protocol
UDP	User Datagram Protocol
UI	User Interface
VOD	Video On Demand
WCG	Wide Color Garmut

Bibliography & References

Alliance for IP Media Solutions http://aimsalliance.org/

SCTE 35 2014 Digital Program Insertion Cueing Message for Cable; Society of Cable Tech Engineers

RTP: A Transport Protocol for Real-Time Applications; IEEE

End-to-end guidelines for phase A implementation http://ultrahdforum.org/resources/phasea-guidelines-description/

